Capstone Project

Stuart McMeechan

March 2018

I. Definition

Project Overview

Determining the health of a business, or an entity within a business, is an important activity for a number of business reasons. For example, analysis of the strength of a business, usually focussed on the balance sheet, will be carried out as standard practice when merging with or acquiring a company. Other examples could include the onboarding of a new supplier or customer, or when carrying out an internal risk review of entities across a global business.

An example of the importance of effectively screening new suppliers relates to the recent collapse of one of the UK’s largest construction services companies. When the company entered administration in 2018, it had £1.5bn of debt and owed up to 30,000 businesses approximately £800m in payments. Could the businesses who are owed money have been aware of the impending collapse of the company? The bankrupt business could either have been their supplier (e.g. playing a key part in their construction project) or customer (e.g. buying parts or people from them).

Problem Statement

Businesses go bankrupt and enter administration regularly. The problem statement is: Given a limited set of financial data typically publicly available, can you predict if a business is in financial distress and likely to collapse?

It is likely that rule-based analysis is common, for example, placing a high risk factor when a business has been selling a high percentage of their fixed assets or if their cash flow is under a specified threshold. However I plan to test whether a machine learning algorithm could produce good predictions.

The anticipated solution will involve capturing historical financial data on companies, with flags to show which ones went bankrupt. This data, following a process of cleansing and feature selection, can be used to train a machine learning algorithm to forecast whether a company is likely to go bankrupt or not, and if it is likely to go bankrupt, will it be in the near future or a number of years away. Different algorithms all suited to multi-class classification problems will be configured, trained and measured, so that the most accurate one can be selected.

The most successful model will be compared to a set benchmark, most likely a random prediction, to check how effective the machine learning approach is.

Metrics

Since only 3-4% of companies in the training dataset go bankrupt, I will use F-score (formula below) to evaluate the models trained.

$$F_{\beta} = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

The F-score, which considers both precision and recall, is required as an evaluation metric since the classification distribution is skewed.

II. Analysis

Data Exploration

A dataset related to companies in Poland going bankrupt was found on the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data), via Kaggle. The dataset contains five files, reflecting the financial metrics of companies from 2007-2013:

File	Data Contains	Labels
Year 1	2008 financial metrics for each company	0 = Company is not bankrupt in 2013 1 = Company is bankrupt by 2013
Year 2	2009 financial metrics for each company	
Year 3	2010 financial metrics for each company	
Year 4	2011 financial metrics for each company	
Year 5	2012 financial metrics for each company	

There are 64 features for each company (example below) but no information on the units of each or if any transformation has been carried out.

BJ	BK	BL	BM
Col62_(short-term_liabilities_*365)_div_sales	Col63_sales_div_short-term_liabilities	Col64_sales_div_fixed_assets	category
82.658	4.4158	7.4277	0
107.35	3.4	60.987	0
134.27	2.7185	5.2078	0
86.435	4.2228	5.5497	0
127.21	2.8692	7.898	0
88.444	4.1269	12.299	0

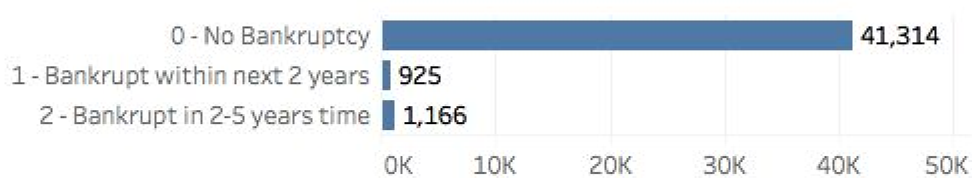
Companies are not identifiable and cannot be linked across the files, so we only know if the company goes bankrupt, relative to 2013. A restriction of this is that, for companies in the **Year 1** file that go bankrupt, we don't know whether they went bankrupt in 2009, 2010, 2011 or 2012.

Since part of the problem statement is to predict whether a company is in imminent danger, I propose that this is a multiclass supervised learning classification problem, with three possible categories for each company: **Predicted to survive** (0), **predicted to go bankrupt within the next 2 years** (1) and **predicted to go bankrupt in 2-5 years** (2).

Target Class

After loading and joining the datasets together, it was noted that the majority of companies do not go bankrupt within the period of analysis (see chart below). Only 5% of companies go bankrupt, which means the dataset is very unbalanced and this is taken into consideration when selecting models and methods of performance scoring.

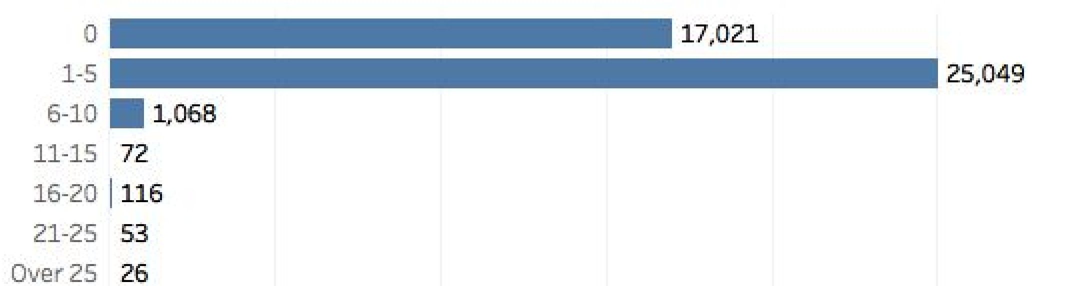
Companies by target class



Features

There are 64 features for each company record and most of these are calculated values, such as *net profit divided by total assets* and *net profit divided by inventory*, all using financial metrics found on a company balance sheet. Data quality is an issue with the dataset. As shown in the chart below, a large number of the records have 1-5 features missing and some have over half of their features missing.

Number of missing features



When considering the bankruptcy status of the companies, it appears the companies that do go bankrupt have a higher number of missing features.

Average number of missing features by class

Class	
0 - No Bankruptcy	2.3%
1 - Bankrupt within next 2..	3.1%
2 - Bankrupt in 2-5 years t..	3.7%

Also, it is much more likely that three specific features are populated when the company does not go bankrupt. For example, for companies that do not go bankrupt, Feature 27 is populated 95% of the time, compared to 62% for companies that go bankrupt within 2-5 years.

% of rows with Feature 27 populated

Class	
0 - No Bankruptcy	95.04%
1 - Bankrupt within next 2 years	70.05%
2 - Bankrupt in 2-5 years time	62.44%

% of rows with Feature 21 populated

Class	
0 - No Bankruptcy	87.32%
1 - Bankrupt within next 2 years	77.08%
2 - Bankrupt in 2-5 years time	65.44%

Algorithms and Techniques

The following supervised learning algorithms were selected to test on the dataset. The algorithms can all be used to solve multi-class supervised learning problems.

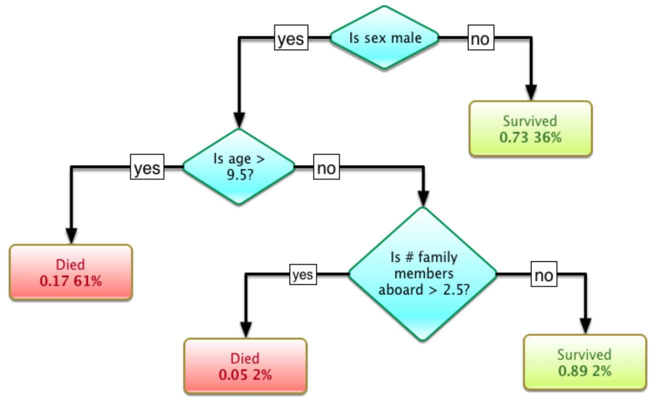
Decision Tree

A decision tree classifier determines the most important features in a training dataset by calculating the information gain if the data was to be split by a given feature. Calculating information gain is based on the concept of entropy, which is defined using the following formula:

$$H(T) = I_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

By iteratively splitting the data on features resulting in the most information gain, the result is a tree containing multiple decisions, with all objects being classified.A decision tree will end when all data has been divided into classes, or it has met the maximum number of 'branches' set in the attributes.

Example decision tree



Example reasons for selecting this type of classifier is that it has good scalability and it can be easy to explain in layman's terms.

Random Forest

A random forest is an ensemble tool which combines observations and features to build multiple decision trees. The class predicted by each decision tree acts as a ‘vote’ and all of the votes are counted to decide on the final predicted target class.

This was selected as an algorithm to test on the bankruptcy data as it can be used as a classifier and also because it is known for given accurate results due to its approach of combining multiple estimators.

When training the random forest algorithm, I plan to leverage parameters such as the ‘maximum depth’, as random forests with too many layers are more likely to overfit on the training data.

KNN

A KNN classifier takes a new object, checks for the k nearest neighbours in the training data and then uses a majority vote system to pick the nearest target class. KNN is a ‘lazy’ algorithm, partly because all training data is still needed when testing and using the algorithm to make predictions.

KNN was chosen to test on the bankruptcy data as it can be used for classification and it is a versatile algorithm used for many different types of predictions, so I was interested to find out how it performed on this dataset.

Logistic regression

Multinomial logistic regression is a classification method for problems with more than two possible target classes.

The logistic regression algorithm in sklearn has a number of parameters available to tune. I plan to test the key ones such as a the different type of classes and associated weights, as well as a limit in the number of iterations.

This type of algorithm was chosen to test as this particular problem has more than two nominal target classes; a problem which multinomial logistic regression is used for.

Gradient boosting

Gradient boosting is a technique which uses a combination of weak prediction models to form a strong prediction. The method is similar to random forests, but gradient boosting can use combinations of any algorithm and uses a weighted average to calculate the final class rather than a simple average.

This type of algorithm was selected as it can be used for multi-class classification problems and it is known to be very efficient. I expect it to perform better than the random forest algorithm I also plan to test.

I plan to test all of the algorithms listed above with the same training data and the performance will be measured using the F1 score.

Benchmark

Before running and testing the supervised learning algorithms on the training data, a benchmark was set using random predictions. This was run using sklearn's DummyClassifier algorithm and the result was an **F1 score of 0.34**.

III. Methodology

Data Preprocessing

Data loading & consolidation

The five source data files were loaded to dataframes. Since the source files only have two target classifications (0 = company survives, 1= company goes bankrupt) a new target class column was added to reflect the three target classifications needed for this analysis. The new target class is based on the source data file, which reflects the timeframe of the companies going bankrupt relative to 2013.

Source File	Target Class Logic
File 1 - 2008 financial metrics for each company	If Class = 1 then New Class = 2 (bankrupt in 2-5 years), Else 0 (company survives)
File 2 - 2009 financial metrics for each company	If Class = 1 then New Class = 2 (bankrupt in 2-5 years), Else 0 (company survives)
File 3 - 2010 financial metrics for each company	If Class = 1 then New Class = 2 (bankrupt in 2-5 years), Else 0 (company survives)
File 4 - 2011 financial metrics for each company	If Class = 1 then New Class = 1 (bankrupt within 2 years), Else 0 (company survives)
File 5 - 2012 financial metrics for each company	If Class = 1 then New Class = 1 (bankrupt within 2 years), Else 0 (company survives)

These five dataframes were then consolidated into a single dataframe:

Feature1	Feature2	Feature65	Target Class
1.9	0.7	4.3	0
0.9	1.1	7.0	0
1.0	0.8	5.0	1
2.5	3.9	3.9	2

Creating new columns

Based on my findings from the initial data exploration, a number of new columns were added to the dataset:

Column	Description
NaN Fields	Number of features that are blank for this company
Is Feature 6 Blank	Flag for whether the specified feature is blank for the company (0,1)
Is Feature 21 Blank	
Is Feature 27 Blank	

Setting NaN values to zero

Following the analysis of the effect of transforming blank values, they are set to zero in the training dataset.

Pre-processing tested but not implemented

A number of other refinements to the dataset were tested, including:

- Removing features with very high numbers of blank values;
- Removing rows with blank values; and
- Setting blank values to the mean instead of zero.

These resulted in a performance decrease, so were not included in the final code.

Implementation

Following the loading and pre-processing of the data, the data was split into training and testing data, with an 80/20 split, using sklearn's train_test_split function.

An initial round of algorithm and testing was carried out using the 'out-of-the-box' algorithms.

```python
# Function for training out of the box algorithms
def getF1score(algo):

    # Train the algorithm
    algo.fit(X_train, y_train)

    # Make predictions on the test data
    predictions = algo.predict(X_test)

    # Return the F1 score
    return f1_score(predictions, y_test, average='macro')
```

```python
# Get F1 scores for each out of the box algorithms
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn import linear_model
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier

clf = RandomForestClassifier(random_state=0)
print ("Random forest:")
print (getF1score(clf))

dtree = tree.DecisionTreeClassifier(random_state=0)
print ("")
print ("Decision tree:")
print (getF1score(dtree))

mul_lr = linear_model.LogisticRegression(random_state=0)
print ("")
print ("Logistic regression:")
print (getF1score(mul_lr))

gbc = GradientBoostingClassifier(random_state=0)
print ("")
print ("Gradient boosting:")
print (getF1score(gbc))

knn = KNeighborsClassifier()
print ("")
print ("KNN:")
print (getF1score(knn))
```

Measuring Performance

As noted in the Data Exploration section, the target classes are heavily skewed, therefore the F-score is used to evaluate the models. Sklearn's F1_score method was used to assess the performance of each of the out-of-the-box algorithms. As shown below, the Gradient Booster algorithm performed the best, with an F1 score of 0.52.

F1 scores for untuned algorithms

Algorithm	
Gradient Booster	0.5225
Decision Tree	0.5037
Random Forest	0.4756
KNN	0.4360
Logistic Regression	0.3391

Refinement

Following the training and testing of the out-of-the-box algorithms, I set out to assess the impact of tuning the key parameters available for each algorithm. The most efficient way of carrying this out was to use sklearn's GridSearchCV, which tests specified parameters and values for a given algorithm and returns the values with the best result.

The function below was written to execute GridSearchCV.

Function for running GridSearchCV

```
from sklearn.grid_search import GridSearchCV

def getbestparameters(params, model_to_tune):

    parameters = params

    grid = GridSearchCV(model_to_tune, param_grid=params, scoring='f1_m
    grid = grid.fit(X_train, y_train)

    return grid.best_estimator_.get_params()
```

For each algorithm a range of parameter values were then passed through the function. The function returned the best performing combination of parameters.

Example of use

```
# Decision Tree - test parameters
dtree = tree.DecisionTreeClassifier(random_state=0)
paramstotry = {'max_depth': range(1,11),
               'min_samples_leaf': range(1,11),
               'min_samples_split': range(2,6)}

best_parameters = getbestparameters(paramstotry, dtree)

for param_name in sorted(paramstotry.keys()):
    print '\t%s: %r' % (param_name, best_parameters[param_name])
```

```
	max_depth: 10
	min_samples_leaf: 1
	min_samples_split: 3
```

Examples of parameters tuned include the following (all used for the decision tree, random forest and gradient booster algorithms):

- max_depth: the maximum depth of the tuned tree. It was important to test difference values for this parameter as if it is too high it's likely the model will overfit.
- min_samples_split: the minimum number of samples needed in order to consider adding another split to the tree.

- min_samples_leaf: the minimum number of samples needed to be at a leaf node.

Impact on Algorithm Performance

After testing the different parameter values, the algorithms were re-trained using these values and re-tested. The updated F1 scores are shown below.

F1 scores for tuned algorithms

Algorithm	Out-of-the-box	Tuned
Gradient Booster	0.5225	0.5489
Decision Tree	0.5037	0.5416
Random Forest	0.4756	0.4620
KNN	0.4360	0.4534
Logistic Regression	0.3391	0.3673

For most algorithms, the parameter tuning resulted in better performance, for example the Decision Tree saw a 7.0% increase in performance and the Logistic Regression algorithm saw a 7.7% increase.

IV. Results

Model Evaluation and Validation

The summary of final F1 scores show that the Gradient Boosting method produced the best predictions on the test data, with an F1 score of 0.549.

Justification

Although the algorithm didn't perform as well as expected, it does perform better than the random selection and it also performs better than the Random Forest algorithm (which is also sometimes used as a benchmark).

When reflecting on the suggested use of the algorithm (to flag if a company, whether a supplier, customer, or other business relationship, is in danger of going bankrupt), this could be a good enough score to justify its use. In a real life situation, a prediction made by this algorithm wouldn't result in an automatic business decision, it would just likely start a process of reviewing company financials in more detail and perhaps speaking with the company to validate or disprove the prediction. If the human follow up of the prediction results in further evidence that the company *is* in danger, then direct action could be taken. This may be to choose not to work with the company if it is a supplier, or perhaps change its payment terms to be 'pay in advance' if it is a customer.

With more time, it would be of interest to assess how non-financial data could impact the algorithm performance, for example *negative mentions in the news/social media*.

V. Conclusion

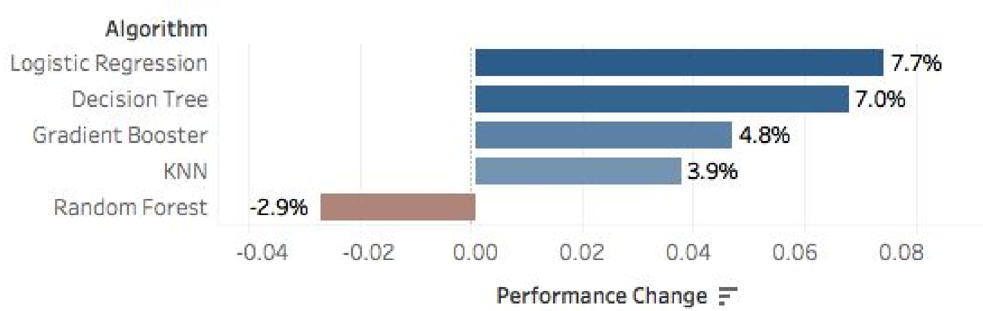
Free-Form Visualization

For the free-form section of the report, I will briefly discuss parameter tuning and feature importance.

Parameter tuning

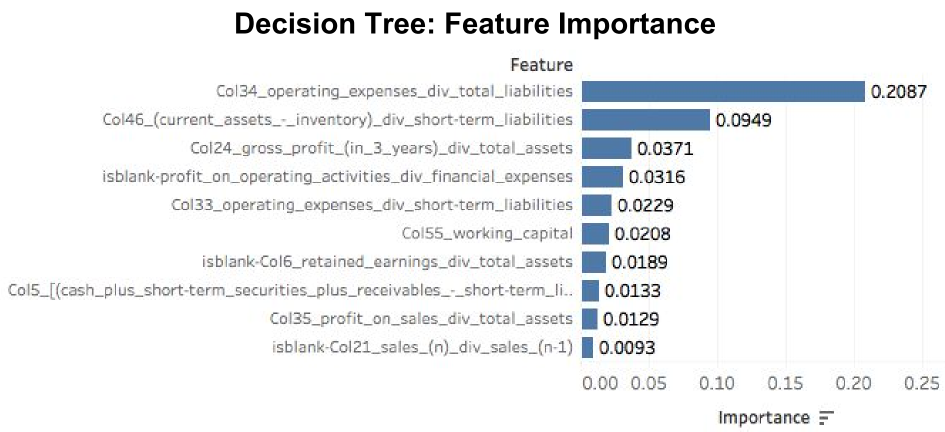
The visualization below summarises the impact that parameter tuning, using GridSearchCV, had on my algorithms. Although it seemed to decrease the performance of the Random Forest, the others all increased, with the Decision Tree and Logistic Regression algorithms increasing by 7% or more. This highlighted the importance of carrying out parameter tuning.

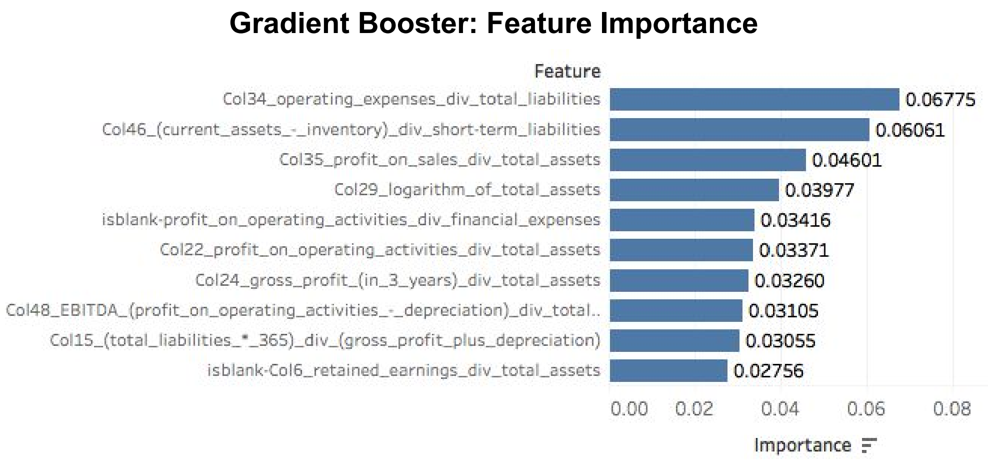
Impact of Parameter Tuning



Feature importance

During the analysis I monitored the feature importance, partly to help my understanding of what the algorithms see as key features. The charts below show the top ten features for the top two performing algorithms.





My observations from the feature importance:

- The top performing algorithms both see features 34 & 46 as the most important features.
- The decision tree has two clear important features, while the gradient booster has a more even spread of importance across a high number of features.
- Two of the features added to the dataset in the data pre-processing stage appear in the most important feature list, which justifies their use.

Reflection

The original problem statement was, given a limited set of financial data typically publicly available, can you predict if a business is in financial distress and likely to collapse. The analysis I carried out involved:

- Exploring the training data (historic financial metrics for companies in Poland, with a tag on those that went bankrupt);

- Processing the training data, including adding new features;
- Training and testing out of the box algorithms;
- Tuning the algorithms with the aim of improving performance;
- Selecting the best algorithm based on the F1 scores; and
- Concluding on the algorithm and its potential value in a real-life setting.

Having reflected on the process of carrying out this project, there are a few areas I found very interesting and there were a few key challenges.

Key interests

I thoroughly enjoyed the exercise overall. An example of a specific interest was, after completing the code to train each algorithm and assess the performance (F1 score), I enjoyed carrying out the iterations of feature cleaning and selection with the aim of increasing the algorithm performance. I look forward to carrying out more research into feature selection methods.

Key challenges and frustrations

I found it frustrating that companies in the training data didn't have a unique identifier. This could have added more depth to the analysis, for example, showing the changing financial metrics as the company neared bankruptcy.

Another challenge was keeping track of the many iterations of re-training I carried out, when using different data processing methods, feature selection etc. I'm interested to research how best to do this.

Improvement

My main improvement consideration is to replace the three target classes with two: **0 - company is likely to go bankrupt**, **1 - company is not likely to go bankrupt**.

My assumption was that I would reach a higher F1 score since there is more training data for the bankruptcy class. This assumption was confirmed, after re-running the same code with only two classes. As shown in the table below, with only two target classes, a decision tree reaches an F1 score of 0.73.

F1 score by algorithm

Algorithm	3 Class	2 Class
Decision Tree	0.5416	0.7315
Gradient Booster	0.5489	0.7262
Random Forest	0.4081	0.7003
KNN	0.4534	0.6624
Logistic Regression	0.3673	0.5340

However from a user point of view, a two class algorithm could be less valuable as there is no indication of whether the company could go bankrupt soon (within the next 2 years). With limited company resources to review businesses in detail, they may prefer a method which can highlight companies likely to go bankrupt within the next 1-2 years.

Sources

http://scikit-learn.org/stable/modules/multiclass.html
https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice
https://en.wikipedia.org/wiki/Multiclass_classification
https://discuss.analyticsvidhya.com/t/what-is-the-fundamental-difference-between-randomforest-and-gradient-boosting-algorithms/2341
http://www.simafore.com/blog/bid/94454/A-simple-explanation-of-how-entropy-fuels-a-decision-tree-model
https://www.tutor2u.net/business/reference/decision-trees
https://en.wikipedia.org/wiki/Decision_tree_learning
https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

Appendix A - Training Data Features

X1 net profit / total assets
X2 total liabilities / total assets
X3 working capital / total assets
X4 current assets / short-term liabilities
X5 [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365
X6 retained earnings / total assets
X7 EBIT / total assets
X8 book value of equity / total liabilities
X9 sales / total assets

X10	equity / total assets
X11	(gross profit + extraordinary items + financial expenses) / total assets
X12	gross profit / short-term liabilities
X13	(gross profit + depreciation) / sales
X14	(gross profit + interest) / total assets
X15	(total liabilities * 365) / (gross profit + depreciation)
X16	(gross profit + depreciation) / total liabilities
X17	total assets / total liabilities
X18	gross profit / total assets
X19	gross profit / sales
X20	(inventory * 365) / sales
X21	sales (n) / sales (n-1)
X22	profit on operating activities / total assets
X23	net profit / sales
X24	gross profit (in 3 years) / total assets
X25	(equity - share capital) / total assets
X26	(net profit + depreciation) / total liabilities
X27	profit on operating activities / financial expenses
X28	working capital / fixed assets
X29	logarithm of total assets
X30	(total liabilities - cash) / sales
X31	(gross profit + interest) / sales
X32	(current liabilities * 365) / cost of products sold
X33	operating expenses / short-term liabilities
X34	operating expenses / total liabilities
X35	profit on sales / total assets
X36	total sales / total assets
X37	(current assets - inventories) / long-term liabilities
X38	constant capital / total assets
X39	profit on sales / sales
X40	(current assets - inventory - receivables) / short-term liabilities
X41	total liabilities / ((profit on operating activities + depreciation) * (12/365))
X42	profit on operating activities / sales
X43	rotation receivables + inventory turnover in days
X44	(receivables * 365) / sales
X45	net profit / inventory
X46	(current assets - inventory) / short-term liabilities
X47	(inventory * 365) / cost of products sold
X48	EBITDA (profit on operating activities - depreciation) / total assets
X49	EBITDA (profit on operating activities - depreciation) / sales
X50	current assets / total liabilities
X51	short-term liabilities / total assets
X52	(short-term liabilities * 365) / cost of products sold)
X53	equity / fixed assets
X54	constant capital / fixed assets
X55	working capital
X56	(sales - cost of products sold) / sales
X57	(current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)

X58	total costs /total sales
X59	long-term liabilities / equity
X60	sales / inventory
X61	sales / receivables
X62	(short-term liabilities *365) / sales
X63	sales / short-term liabilities
X64	sales / fixed assets